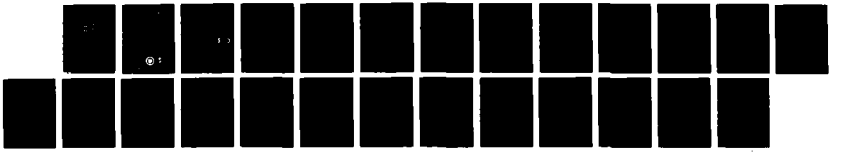
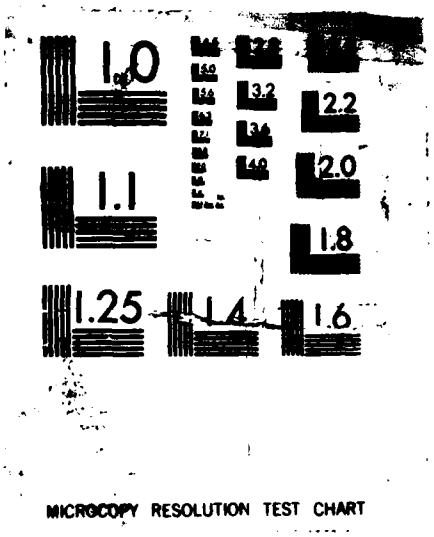
AD-A182 111 ITERATIVE SOLUTIONS OF SPARSE LINEAR SYSTEMS ON SYSTOLIC ARRAYS(U) PITTSBURGH UNIV PA INST FOR COMPUTATIONAL MATHEMATICS AND APPLICATIONS R MELHEM

1/1

UNCLASSIFIED MAR 87 ICMA-87-105 N00014-85-K-0339 F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART

(14)

INSTITUTE FOR COMPUTATIONAL MATHEMATICS AND APPLICATIONS

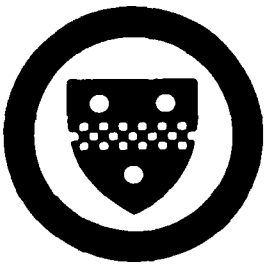
Technical Report ICMA-87-105 March, 1987

Iterative Solutions of Sparse Linear Systems on Systolic Arrays*)

by

Department of Mathematics and Statistics

University of Pittsburgh



(14)

Technical Report ICMA-87-105 March, 1987

Iterative Solutions of Sparse Linear Systems on Systolic Arrays*)

by

Rami Melhem

Department of Computer Science

and

Institute for Computational Mathematics and Applications
Department of Mathematics and Statistics
University of Pittsburgh
Pittsburgh, PA 15260

*) This work is, in part supported under ONR contract N00014-85-K-0339 and Air Force contract AFOSR-84-0131.

This paper will be presented at the 16th International Conference on Parallel Processing, August 1987.

-A-

ITERATIVE SOLUTION OF SPARSE LINEAR SYSTEMS ON SYSTOLIC ARRAYS

Rami Melhem

The University of Pittsburgh

Pittsburgh, PA 15260

ABSTRACT

The idea of grouping the non-zero elements of a sparse matrix into few stripes that are almost parallel is applied to the design of a systolic accelerator for sparse matrix operations. This accelerator is, then, integrated into a complete systolic system for the solution of large sparse linear systems of equations. The design demonstrates that the application of systolic arrays is not limited to regular computations, and that computationally irregular problems may be solved on systolic networks if local storage is provided in each systolic cell for buffering the irregularity in the data movement and for absorbing the irregularity in the computation.

1. INTRODUCTION

In [6] and [7], stripe structures of matrices were introduced as a means for the inclusion of all the non-zero elements of a sparse matrix into a regular pattern which is suitable for parallel computation. This stripe approach have proved to be useful for the manipulation of sparse matrices on data-driven networks. It is especially attractive for the type of matrices that result from finite element analysis. More specifically, the number of stripes, π, in finite element matrices is small and depends only on the method used for the discretization. That is, π is independent of the size of the problem. Algorithms for the generation of stripe structures for sparse matrices, in general, and for finite element matrices, in particular, are given in [6] and [8], respectively..

However, stripes, as defined in [6], are not regular enough to allow for the manipulation of sparse matrices on systolic networks. More specifically, a particular stripe may, itself, be sparse. In data driven networks, where the operation of each cell is initiated by the availability of data, the sparsity within a stripe may cause execution delays, but does not affect the correctness of the computation. On the other hand, systolic arrays, which are globally synchronized, are characterized by the property that the location of each data item at any time may, and should, be determined before the beginning of execution. This is not possible if stripes are sparse.

Hence, for each sparse stripe S, we introduce, in Section 2, a non sparse structure which we call the complement of S. Using the complements of the stripes, it is possible to design linear systolic arrays for the multiplication of a vector by a striped matrix. These arrays are presented in Section 3, along with an analysis which determines the exact timing of the input data required for the correctness of the computation.

In Section 4, the last cell in the matrix/vector multiplication array is modified to feed data back into the array and to obtain a network for the solution of triangular linear systems. The feed back, however, creates a data dependence which may lead to incorrect computations. A condition which guarantees the correctness of the computation is formulated

in terms of the separation between the stripes of the matrix and the rate at which the input data is supplied to the network.

Matrix/vector multiplication and the solution of triangular linear systems are the only $O(n^2)$ operations in each iteration of the preconditioned conjugate gradient method (PCCG), which is one of the most efficient techniques used for the iterative solution of large sparse linear systems. Hence, these two matrix operations should be the primary target for acceleration in any parallel system used for the solution of large linear systems. However, the application of the PCCG involves also some $O(n)$ operations, as for example vector addition and inner product computations. These $O(n)$ operations may not be ignored because they may become the bottle-neck of the parallel system (see [4] for example). A complete systolic system which accelerates both $O(n^2)$ and $O(n)$ operations in the PCCG method is described in Section 5.

2. STRIPE STRUCTURES AND THEIR COMPLEMENTS

In [6], a stripes S in a matrix $A = \{a_{i,j} : i,j=1,....n\}$ is defined to be a set of positions in A which contains at most one position for each row. More specifically,

$$S = \{(i,\sigma(i)) : i \in i\_dom(S), \quad \text{and} \quad \sigma(i) < \sigma(l) \text{ for } i < l\} \tag{1}$$

where $i\_dom(S)$ is a subset of $\{1, \cdots n\}$ and σ is an increasing function which associates a column index $\sigma(i)$ with row i. An alternative definition of a stripe may be given in terms of a function μ which associates a row index $\mu(j)$ with column j. That is

$$S = \{(\mu(j),j) : j \in j\_dom(S) \text{ ,and } \mu(j) < \mu(m) \text{ for } j < m\} \tag{2}$$

where $j\_dom(S)$ is a subset of $\{1, \cdots ,n\}$. Clearly, if $i \in i\_dom(S)$, then $\sigma(i) \in j\_dom(S)$ and $i = \mu(\sigma(i))$. That is $\mu = \sigma^{-1}$.

Given two stripes $S_1=\{(i,\sigma_1(i))\}$ and $S_2=\{(i,\sigma_2(i))\}$ the ordering relation $<$ may be defined such that $S_1 < S_2$ if $\sigma_1(i) < \sigma_2(j)$ for $i < j$. With this, a stripe structure of A is a set of stripes $\Sigma_A = \{S_1,....S_\pi\}$ such that $S_1<S_2<...<S_\pi$ and $S_1 \bigcup \cdots \bigcup S_\pi$ contains all the positions of the nonzero elements in A. That is, if $(i,j) \notin S_1 \bigcup \cdots \bigcup S_\pi$, then

$a_{i,j}=0$. π is called the stripe count of Σ_A. As an example, we denote in Figure 1 the zero and nonzero elements of a matrix by '.' and 'x', respectively, and we show a possible stripe structure for that matrix. Note that the positions included in the stripes are enclosed in circles.

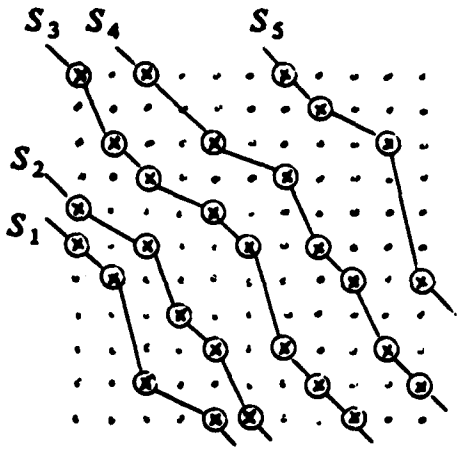


Figure 1 - A stripe structure

From the definitions (1) and (2), it is not essential that S contains a position for every row $i=1,...,n$ or for every column $j=1,...,n$, and thus, stripes may be sparse. Although stripes which are sparse introduce some irregularity in the stripe structure of matrices, it is shown in [6] that linearly connected arrays of cells may be used effectively for the manipulation of striped matrices, provided that data-driven synchronization is used. In other words, it is possible to cope with the sparsity within a stripe, if the operation of each cell is initiated by the availability of data.

On the other hand, if the array is synchronized by a global clock, then each cell is expected to receive some data every clock cycle. Hence, sparse stripes should be augmented such that a data item is presented to the network each clock cycle. One way of augmenting sparse stripes is to include in S a position for each row of A. In this case, the augmented stripe is called the row complement of S and is defined by

$$\bar{S}^r = \{(i, \bar{\sigma}(i)):\ i = 1,...,n\}$$

where

$$\bar{\sigma}(i) = \sigma(i) \qquad \text{if } i \in i\_dom(S) \tag{3.a}$$

and

$$\bar{\sigma}(i) \geqslant \bar{\sigma}(i-1) \qquad i=2,\ldots,n \tag{3.b}$$

Note that $\bar{S}^r$ is not a stripe because $\bar{\sigma}$ is not strictly increasing. In fact, given S, it may be impossible to augment σ such that $\bar{\sigma}$ is strictly increasing. Although conditions of the form given by (3) may be shown to ensure the correct operation of systolic networks on striped matrices, it should be clear that (3.a/b) do not define $\bar{S}^r$ uniquely, and that a unique row complement of S may be only obtained if $\bar{\sigma}$ is defined in a more restrictive manner. In this paper, we adopt the following unique definition of $\bar{\sigma}$:

$$\bar{\sigma}(i) = \begin{cases} \sigma(i) & \text{if } i_n \leqslant i \leqslant i_x \text{ and } i \in i\_dom(S) \\ \bar{\sigma}(i+1) & \text{if } i_n \leqslant i \leqslant i_x \text{ and } i \notin i\_dom(S) \\ \sigma(i_n) - (i_n - i) & \text{if } i < i_n \\ \sigma(i_x) + (i - i_x) & \text{if } i > i_x \end{cases} \tag{4}$$

where $i_x = \max\{i : i \in i\_dom(S)\}$ and $i_n = \min\{i : i \in i\_dom(S)\}$.

Similarly, the column complement of S may be defined by adding to S one position for each column $j \notin j\_dom(S)$. More specifically,

$$\bar{S}^c = \{(\bar{\mu}(j), j) : j = 1,\ldots,n\}$$

where

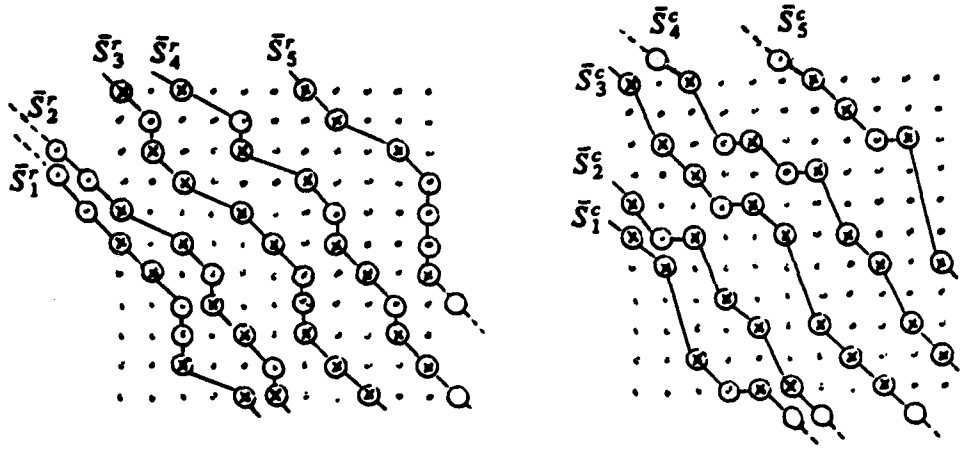
$$\bar{\mu}(j) = \mu(j) \qquad \text{if } j \in j\_dom(S) \tag{5.a}$$

and

$$\bar{\mu}(j) \geqslant \bar{\mu}(j-1) \qquad j=2,\ldots,n. \tag{5.b}$$

and a unique definition of $\bar{\mu}$ may be given in a way analogous to (4).

Given a stripe structure $\Sigma_A = \{S_1,\ldots,S_\pi\}$, the row and column complements of Σ_A are defined, respectively by $\bar{\Sigma}_A^r = \{\bar{S}_1^r, \ldots, \bar{S}_\pi^r\}$ and $\bar{\Sigma}_A^c = \{\bar{S}_1^c, \ldots, \bar{S}_\pi^c\}$. In Figure 2 we show the complements of the structure of Figure 1. Note that any position (i, j) in $\bar{\Sigma}_A^r$ and $\bar{\Sigma}_A^c$ with i or j not in $\{1, \cdots n\}$ is assumed to contain a zero.



(a) Row complement (b) Column complement

Fig 2 - The complements of the structure of Fig 1

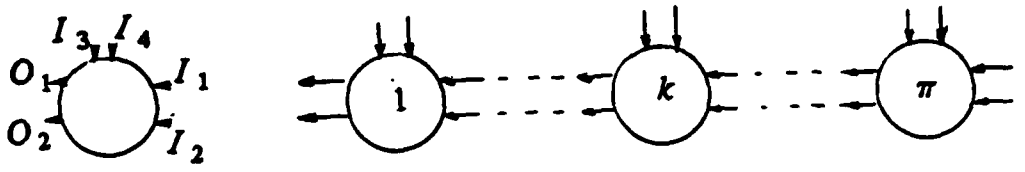
3. SYSTOLIC MULTIPLICATION OF A VECTOR BY A STRIPED MATRIX

3.1. Using column complemented stripe structures.

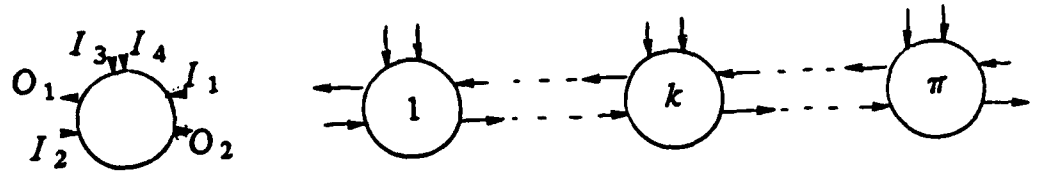
In this section, we assume that a stripe structure $\Sigma_A = \{S_1, \cdots S_\pi\}$ is given for a matrix A, and we present a systolic algorithm for the multiplication of any vector x by A using π linearly connected systolic cells (see Figure 3). Each systolic cell k has four input ports (denoted I_1 - I_4) and two output ports (denoted O_1 and O_2). It also contains storage for an n-dimensional array SX_k which is initialized to zero. In brief, the elements of the stripe S_k are multiplied, in cell k, by the corresponding elements of x and the results are stored in SX_k. The partial sums are then accumulated across the cells $1, \cdots, \pi$.

In order to describe the algorithm, we let T be the time defined by the global clock (number of cycles since the start of operation), and we assume that the elements of the vector x are applied to port I_1 of cell π starting at time $T=1$, and that each cell transmits the content of I_1 to O_1 in one cycle. We also assume that each cell k executes $\pi-k$ trivial cycles before it starts receiving actual data and performing useful operations. for simplicity, we let $t_k = T-(\pi-k)$ be a local time defined at cell k.

At any given local cycle t_k of cell k, the element $a_{\bar{\mu}_k(t_k)J_k}$ of the complemented stripe $\bar{S}_k^c$ is supplied to I_3, and the row index $\bar{\mu}_k(t_k)$ is supplied to I_4. During the same cycle, the



(a) Unidirectional data flow



(b) Bidirectional data flow
Fig 3 - Systolic striped matrix/vector multiplication

corresponding element x_{t_k} should arrive at I_1, thus allowing for the formation of the product term $a_{\bar{\mu}_k(t_k),s_k} * x_{t_k}$. This term is then stored in $SX_k[\bar{\mu}_k(t_k)]$. Here, we note that if $t_k \in j\_dom(S_k)$, then $\bar{\mu}_k(t_k) = \mu_k(t_k)$, and hence, $a_{\bar{\mu}_k(t_k),s_k} * x_{t_k} = a_{i,\sigma_k(i)} * x_{\sigma_k(i)}$, where $i = \mu_k(t_k)$.

The elements $y_1, \cdots, y_n$ of the y data stream ,initialized to zero, are propagated through the network either in the same direction as the x data stream (see Fig 3a), or in the opposite direction (see Fig 3b). In either case, when an element y_i arrives at cell k, it picks up the corresponding term stored in $SX_k[i]$. But $SX_k[i] = a_{i,\sigma_k(i)} * x_{\sigma_k(i)}$ if S_k contains a position at row i, and $SX_k[i]=0$ otherwise. Hence, for any specific i, y_i visitsall of the π cells and accumulates the sum of the terms $a_{i,j}\ x_j$, for which $(i,j) \in S_k$ for some k. Given that $a_{i,j}=0$ if there is no k such that $(i,j) \in S_k$, we conclude that $y_i = \sum_{j=1}^{n} a_{i,j}\ x_j$, the i^{th} element of the product $A\ x$.

Both the x and y data streams may flow in the network simultaneously. However, it is important to ensure that an element y_i will not arrive at a cell k before the term

$a_{i,\sigma_k(i)} * x_{\sigma_k(i)}$ is computed and stored in $SX_k[i]$. With this in mind, the operation of cell k in the network may be described by the algorithm of Fig 4, in which the y stream is assumed to arrive at cell k lagging behind the x stream by δ_k cycles. For simplicity, we assume that storage is allocated (and initialized to zero) in each cell for $SX_k[i]$, $i = -\delta_k, \cdots, 0$, and that $x_i = 0$ for $i = n+1, \cdots, n+\delta_k$.

Starting at time $T = \pi - k + 1$ DO

FOR cycles $t_k = 1, \cdots, n+\delta_k$ DO

1) Read x_{t_k} from I_1 into ξ and $y_{t_k-\delta_k}$ from I_2 into η.

2) Read $a_{\bar{\mu}_k(t_k),t_k}$ from I_3 into α and $\bar{\mu}_k(t_k)$ from I_4 into λ.

3) $SX_k[\lambda] \leftarrow \alpha * \xi$

4) $\eta \leftarrow \eta + SX_k[t_k - \delta_k]$

5) Write ξ and η on O_1 and O_2, respectively.

Fig 4 - The operation of cell k

The next goal is the determination of the value of δ_k which ensures that $SX_k[t_k - \delta_k]$ is computed in some cycle $t_k' < t_k$. In order to be able to apply our results on two leveled pipelined systolic machines [1], we assume that the multiplication and the addition in each cell are performed on pipelined units with p^* and p^+ stages, respectively. In this case, the results of step 3 in Fig 4 is stored in SX_k at cycle $t_k + p^*$. This means that at cycle t_k, the values of $SX_k[1], \cdots, SX_k[\bar{\mu}_k(t_k)] - p^*$ are already computed and thus we must have

$$t_k - \delta_k \leqslant \bar{\mu}_k(t_k) - p^*$$

Which is satisfied if

$$\delta_k \geqslant j - \mu_k(j) + p^* \qquad \text{for } any \quad j \in j\_dom(S_k) \tag{6.a}$$

Equation (6.a) specifies the minimum value of δ_k required in cell k to ensure correct operation. However, the elements of the y data stream flow in the cells of the network in an orderly manner, which implies some relation between the values δ_k, $k = 1, \ldots, \pi$.

In order to be more specific, we consider the case of bidirectional data flow (Fig 3b). If y_l and y_m are at cells k and $k+1$, respectively, during the same global cycle T, then $l = t_k - \delta_k = T - \pi + k - \delta_k$ and $m = t_{k+1} - \delta_{k+1} = T - \pi + k + 1 - \delta_{k+1}$. But the y stream has to travel through the addition pipeline units, and hence $l = m + p^+$, which gives

$$\delta_{k+1} = \delta_k + p^+ + 1 \tag{6.b}$$

Hence, δ_k depends on δ_1, which, in turns, depends on the time, $\tau_{start\ y}$, at which the input $y_1, y_2, \cdots$ is initiated at port I_2 of cell 1. More specifically, $\tau_{start\ y}$ is the global time T corresponding to the local time $t_1 = \delta_1 + 1$ at cell 1. This gives

$$\delta_1 = \tau_{start\ y} - \pi \tag{6.c}$$

Clearly, it is possible to satisfy (6.a) by taking $\tau_{start\ y} = n + \pi + p^*$, which means that the y stream is applied into the network after the x stream exits it, thus ensuring that $SX_k[i]$ will contain the correct values upon the arrival of y_i. This two-phase approach, however, has two disadvantages, namely, 1) if each cell is capable of performing both a multiplication and an addition in each cycle (as is the case with the Warp [1]), then the cells are under-utilized because only a multiplication is performed in the first phase, and only an addition is performed in the second phase, and 2) the two phase approach is not applicable to the solution of triangular linear systems, in which the values in the x stream depends on r^+ results obtained in the y stream.

Fortunately, we do not have to wait until the x stream exits the network before we input the y stream. More specifically, it is straight forward to check that the condition (6.a) is satisfied if

$$\tau_{start\ y} = B_2 + p^* + \pi \tag{7}$$

where B_2 is the upper band-width of the matrix A, which is necessarily larger than $j - i$ for any $a_{i,j} \neq 0$, and thus, larger than $j - \mu_k(j)$ for any $j = 1, \ldots, n$ and $k = 1, \ldots, \pi$. Moreover, it may be shown that $\tau_{start\ y}$ given by (7) is the minimum starting time for the y stream which will always guarantee the correct operation of the network. With this

starting time, any specific y_i appears on port O_2 of cell π at time $i+B_2+(p^+ +1)\pi+p^*$. Hence, the matrix/vector multiplication may be computed in $n+B_2+(p^+ +1)\pi+p^*$ systolic cycles.

The same type of analysis may be applied to the case where the y data stream flows in the same direction as the x stream. In this case, however, equation (6.b) is replaced by $\delta_k = \delta_{k+1}+p^+ +1$, and equation (6.c) is replaced by $\delta_\pi = \tau_{start,y}-1$. The minimum starting time for the y stream and the execution time of the network may then be found to be $B_2+p^* +1$ and $n+B_2+p^+\pi+p^* +1$ respectively.

Finally, it should be noted that at any particular cycle t_k of cell k, only the locations $SX_k[\bar{\mu}_k[t_k], \ldots, SX_k[t_k-\delta_k]$ of SX_k may be occupied. Hence, the n dimensional array SX_k may be replaced by a circular buffer of length $\max_{k,j}\{\mu_k(j)-j+\delta_k\} = B_1+B_2+2p^*$, where B_1 is the lower band-width of A. It satisfies $B_1 \geq \max_{k,j}\{\mu_k(j)-j\}$.

3.2. The multiplication of x by A^T

The exact same networks described in the previous section may be used for the multiplication of x by the transpose of the matrix A. Namely, if the row complement of S_k rather than its column complement is supplied to cell k in the networks of Fig 3, then the y streams will accumulate the elements of the product vector $A^T x$.

In order to be more specific, we assume that each cell k in the networks of Fig 3 executes the algorithm of Fig 4 after the replacement of step 2 by

2) Read $a_{t_k,\sigma_k(t_k)}$ from I_3 into α and $\bar{\sigma}_k(t_k)$ from I_4 into λ.

With this replacement, the term $a_{t_k,\sigma_k(t_k)} * x_{t_k}$ is computed at cycle t_k and stored in $SX_k[\bar{\sigma}_k(t_k)]$. But, if $t_k \in i\_dom(S_k)$, then $\bar{\sigma}_k(t_k) = \sigma_k(t_k)$ and $a_{t_k,\sigma_k(t_k)} * x_{t_k} = a_{\mu_k(j),j} * x_{\mu_k(j)}$, where $j = \sigma_k(t_k)$. Hence, for any $j=1,\cdots,n$, $SX_k[j]$ contains $a_{\mu_k(j),j} * x_{\mu_k(j)}$ if $(\mu(j),j) \in S_k$, and contains zero otherwise.

Now, each element y_j in the y data stream visits every cell k and picks up from it the content of $SX_k[j]$. Upon exiting the network, y_j, thus, contains the sum of the terms $a_{i,j}\ x_i$ for which $(i,j) \epsilon S_k$ for some k. But, $a_{i,j}=0$ if (i,j) does not belong to any stripe. Hence, $y_j = \sum_{i=1}^{n} a_{i,j}\ x_i$, the j^{th} element of $A^T\ x$.

The value of the delay factor δ_k which ensures the correct operation may be obtained by an analysis identical to the one described in Section 3.1. This analysis gives a condition on δ_k similar to (6.a). Namely

$$\delta_k \geqslant i - \sigma(i) + p^* \qquad \text{for } any \quad i \in i\_dom(S_k) \tag{8}$$

Assuming that the y stream flows in the direction opposite to that of the x stream, then both (6.b) and (6.c) hold. The minimum starting time for the y data stream is, then, found to be

$$\bar{\tau}_{start\ y} = B_1 + p^* + \pi \tag{9}$$

where B_1 is the lower band-width of the matrix A. In this case, the execution of the network requires $n+B_1+\gamma$ cycles, where $\gamma=(p^*+1)\pi+p^*$.

3.3. The case of symmetric matrices

Let H be a symmetric matrix which may be decomposed into $A + D + A^T$, where A is a strictly lower triangular matrix and D is a diagonal matrix. Let also $\Sigma_A = \{S_1, \cdots S_{\pi-1}\}$ be a stripe structure for A and $\Sigma_D = \{S_\pi\}$ be the stripe structure of D, where S_π contains all the diagonal positions (i,i), $i=1,....n$. Clearly, the matrix H has $2\pi-1$ stripes, and hence, the multiplication of any vector x by H may be performed on a systolic network with $2\pi-1$ cells in approximately $n+B_2$ cycles, as described in the previous sections.

However, the goal of this paper is to use systolic networks in the iterative solution of linear systems. This involves, besides matrix/vector multiplication, the solution of triangular linear systems. In Section 3, it is shown that the solution of triangular systems may be

performed on a systolic network composed of π cells. Hence, if the multiplication $y = Hx$ may also be performed using only π cells, then the same systolic network may be used for both operations, thus increasing the utilization of the network, and the efficiency of the entire solution process.

The reduction of the number of cells used in the multiplication $y = H\ x$ may be accomplished by first computing the partial result vector $z = (A + D)\ x$, and then computing $y = z + A^T\ x$. Given that the stripe count of A is $\pi - 1$, and that of D is one, then, the two operations may be performed, sequentially, on a network of π cells. This, however, requires that the partial result vector z be stored outside the network.

It is possible to avoid the external storage of z if the two operations $(A + D)x$ and $A^T x$ are interleaved. In this case, the internal storage SX_k in cell k, $1 \leq k < \pi$, may be used to hold temporarily the product of stripe S_k by x (generated during the multiplication $(A + D)x$), and then to accumulate the product of the transpose of S_k by x (generated during the multiplication $A^T x$). Clearly, cell π is only responsible for the multiplication of S_π by x and hence no interleaved operation is needed for that cell.

A more precise description of the operation of a cell k, $1 \leq k < \pi$, is given in Fig 5. Note that each execution of the body of the FOR loop corresponds to two systolic cycles, where in any specific cycle, one addition, one multiplication, and at most one read/write from/to each port, may be performed.

It is important to ensure that the content of a location $SX_k[\tilde{\lambda}]$, which is computed at step 2.2 in any given cycle, is not overwritten by the execution of step 1.3 in any future cycle. In other words, for any given t_k, $\lambda = \bar{\sigma}_k(t_k)$ should be smaller than $\tilde{\lambda} = \bar{\mu}_k(t_k)$. This condition is always satisfied for any lower triangular matrix A.

As clear from the algorithm of Fig 5, the x and y streams travel in the network at a speed of one cell every two cycles. The value of the delay δ_k should satisfy both conditions (6.a) and (7). Assuming that the two data streams flow in opposite directions, then the minimum starting time for the y data stream is the largest of $\tau_{start\ y}$ and $\bar{\tau}_{start\ y}$ given

Starting at time $T = \pi - k + 1$ DO

FOR $t_k = 1, \cdots, n + \delta_k$ DO

Cycle 1: 1.1) Read x_{t_k} from I_1 into ξ and $y_{t_k - \delta_k}$ from I_2 into η.

1.2) Read $a_{t_k, \sigma_k(t_k)}$ from I_3 into α and $\overline{\sigma}_k(t_k)$ from I_4 into λ.

1.3) $SX_k[\lambda] \leftarrow \alpha * \xi$

1.4) $\eta \leftarrow\!- \eta + SX_k[t_k - \delta_k]$

Cycle 2 : 2.1) Read $a_{\overline{\mu}_k(t_k), t_k}$ from I_3 into α and $\overline{\mu}_k(t_k)$ from I_4 into $\tilde{\lambda}$.

2.2) $SX_k[\tilde{\lambda}] \leftarrow SX_k[\tilde{\lambda}] + \alpha * \xi$

2.3) Write ξ and η on O_1 and O_2, respectively.

Fig 5 - the multiplication of a vector by a symmetric matrix

by (7) and (9), respectively. However, A is a lower triangular matrix for which B_2=0. Hence, the minimum starting time for the network is given by (9), and the computation of Hx terminates in $2(n + B_1 + \gamma)$ cycles, where $\gamma = (p^+ + 1)\pi + p^*$ is a constant which depends on the number of cells and the architecture of each cell.

4. THE SOLUTION OF TRIANGULAR LINEAR SYSTEMS

In this section, we consider lower triangular linear systems of the form

$$(A + D)x = b$$

where, as in Section 3.3, A is strictly lower triangular with stripe structure $\Sigma_A = \{S_1, \ldots, S_{\pi-1}\}$, and D is diagonal with stripe structure $\Sigma_D = \{S_\pi\}$. The solution of such systems may be computed on any of the two systolic networks shown in Figure 3. For simplicity, we start by assuming that the network has bidirectional data flow (Fig 3b).

The operation of the network is similar to that of the original forward substitution network of Kung and Leiserson [5]. However, the same techniques used in Section 3.1 are applied such that each cell deals with a stripe rather than a diagonal. More specifically, the first $\pi - 1$ cells of the network, namely cells $k = 1, \cdots, \pi - 1$, perform the matrix/vector

multiplication $y = Ax$ starting at time τ_x. That is the elements of x are supplied to port I_1 of cell $\pi-1$ starting at time τ_x. Subsequent analysis will show that $\tau_x = p^+ + p^* + 1$.

Hence, the operation of each cell k, $1 \leq k \leq \pi-1$, is described by the same algorithm of Fig 4, except that execution starts at time $T = \pi - k - 1 + \tau_x$. This change in the starting time does not affect the conditions (6.a) and (6.b) that should be satisfied in order to ensure correct operation and data flow.

The last cell in the network, namely cell π, is responsible for recursively computing the i^{th} element of x from the i^{th} element of $y = Ax$. Assuming that the elements of the right hand side vector b are applied to port I_1 of cell π starting at time $T = 1$, and noting that $\mu_\pi(i) = i$, we may describe the operation of cell π by the following Algorithm:

FOR cycles $t_\pi = 1, \cdots, n$ DO /* Here $T = t_\pi$ */

1) Read $(1 / a_{t_\pi, t_\pi})$ from I_3 into α

2) Read b_{t_π} from I_1 into ξ, and y_{t_π} from I_2 into η.

3) $\xi \leftarrow (\xi - \eta) * \alpha$

4) Write ξ on O_1 and O_2. /* O_2 is considered the output of the network */

First, we note that we avoided the division operation in cell π by supplying to the cell the reciprocals of the diagonal elements, which, in this case, should be computed outside the network (by the host for example). Relieving cell π from performing the division operation allows the execution rate of all the cells in the network to be equal, but increases the load on the host. However, during the iterative solution of linear systems the same triangular system is solved in each iteration for a different vector b. Given that we usually iterate hundreds of times, the one time increase in the load of the host is justifyable.

Assuming that multiplication and subtraction (equivalent to addition) in cell π are pipelined, it is clear that the value of x_i appears on port O_2 of cell π (and thus on I_1 of cell $\pi-1$) at global cycle $T = p^+ + p^* + i$. During that same global cycle, $y_{p^+ + 2p^* + i}$ should be at port I_2 of cell $\pi-1$. Hence, from the algorithm of Fig 4, we find that the value of $\delta_{\pi-1}$ at cell $\pi-1$ is equal to $-(p^* + 2p^+)$, and thus, from (6.b)

$$\delta_k = -(p^*-2+(\pi-k+1)(p^+ +1)) \qquad k=1,\cdots,\pi-1 \tag{10}$$

In other words, δ_k, $k=1,\ldots,\pi-1$, are completely determined by the data flow. However, condition (6.a) should still be satisfied in order to ensure the correct operation of cells $1,\cdots,\pi-1$. With (10) this condition becomes:

$$\mu_k(j)-j \geq 2(p^*-1)+(\pi-k+1)(p^+ +1) \qquad k=1,\cdots,\pi-1 \tag{11}$$

which basically specify that stripe S_k should be separated from the diagonal by at least the right side of the inequality. Hence, the network operates correctly only if the stripes of the input matrix satisfy condition (11). In this case, the last element of the solution vector x is computed at global time $T=\tau_x+n$, that is the network terminates its execution after $p^+ + p^* + 1 + n$ cycles.

For example, assume that both the multiplication and the addition are performed on 5-stage pipelined units, that is $p^+ = p^+ = 5$. Hence, equation (11) indicates that the first lower stripe, namely $S_{\pi-1}$ should be away from the diagonal by at least 20 positions, the second lower stripe, $S_{\pi-2}$ by at least 26 positions, and so on. Multicolor numbering techniques are available [8, 9], which rearrange the rows and columns of the matrix such that to satisfy this type of stripe separation.

The stripe separation condition (11) may be relaxed if the data in the input streams to the network are spread appropriately. More specifically, if the data at any input port are applied at the rate of one data item every θ cycles, rather than every cycle, then it may be shown that $\theta\,\delta_k$ will appear in the left side of equation (10), and thus the condition (11) becomes

$$\theta(\mu_k(j)-j) \geq 2(p^*-1)+(\pi-k+1)(p^+ +1) \qquad k=1,\cdots,\pi-1 \tag{12}$$

which may always be satisfied by the proper choice of θ. Clearly, the execution time in this case increases to $n\,\theta + p^* + p^+ + 1$ cycles.

Although we considered, so far, only networks with bidirectional data flow, it is

equally possible to solve triangular systems on networks with unidirectional data flow. For this, the y data stream is initiated at port I_2 of cell $w-1$, and the result of Ax is fed back from port O_2 of cell 1 to port I_2 of cell w (see Fig 6). The operation of each cell in the network remains the same, but the value of δ_k may be found to be $\delta_k = -(p^* - (k+1)p^+)$. Correct operation of the network, in this case, requires that the stripes of the input matrix satisfy $\mu_k(j) - j \geq (k+1)p^+$ rather that (11).

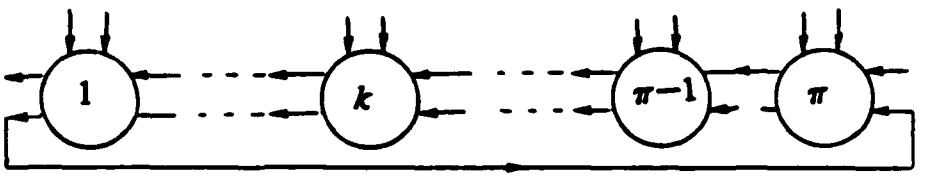


Fig 6 - Unidirectional data flow with feed back

Finally, it should be noted that the solution of upper triangular systems of the form $(A^T + D)x = b$ are very similar to the solution of lower triangular systems. Hence, the same networks may be used and the same analysis techniques may be applied.

5. APPLICATION TO THE PRECONDITIONED CONJUGATE GRADIENT METHOD

The preconditioned conjugate gradient method (PCCG) [2] is one of the best known iterative method for the solution of large sparse linear systems of the form $Hx = f$, where H is an $n \times n$ symmetric, positive definite matrix. Each iteration of the PCCG involves the multiplication of a vector by H and the solution of two triangular systems of the forms $(A + D)x = b$ and $(A^T + D)h = x$, where A is a strictly lower triangular matrix derived from H. Denoting by $\langle x, y \rangle$ the inner product of two vectors x and y, the PCCG may be described as follows:

Choose an initial guess x_0.

$r_0 = f - Hx_0$; $h = M^{-1} r_0$; $\gamma_0 = \langle r_0, h \rangle$; $\beta = 0$

Repeat for $i = 0, \cdots$ until $\gamma_i \leq \epsilon$, where ϵ is an acceptable error

1) 1.1) $p_i = h + \beta p_{i-1}$

1.2) $y = H p_i$

1.3) $\alpha = \langle y, p_i \rangle$.

2) 2.1) $\alpha = \frac{\gamma_i}{\alpha}$,

3) 3.1) $x_{i+1} = x_i + \alpha p_i$

3.2) $r_{i+1} = r_i - \alpha y$

3.3) Solve $(A+D)\, b = r_{i+1}$

4) 4.1) Solve $(A^T+D)\, h = b$,

4.2) $\gamma_{i+1} = \langle r_{i+1}, h \rangle$

5) 5.1) $\beta = \frac{\gamma_{i+1}}{\gamma_i}$.

If H is irregularly sparse, then A is usually chosen such that $(A+D+A^T)$ has the same sparsity structure as H, and thus the same stripe structure. Let $\Sigma_H = \{S_1, \cdots, S_\pi, \cdots S_{2\pi-1}\}$ be a stripe structure for H such that S_π is the diagonal $\{(i,i) : i = 1,\ldots,n\}$. Hence, $\Sigma_A = \{S_1, \cdots, S_{\pi-1}\}$.

In Fig. 7, we show a complete systolic system for the execution of the PCCG. The rectangular block labeled ARRAY in the figure is a systolic array consisting of π cells. This array is used for the execution of steps 1.2, 3.3 and 4.1 in each iteration of the PCCG. Each cell in ARRAY, however, is slightly different from the cell shown in Fig 3. More specifically, the input ports I_3 and I_4 in each cell in Fig 3 are omitted in ARRAY, and the data which is supplied on these ports, namely the elements of the stripes of H and A, is stored in local memories. In other words, each cell k in ARRAY has enough memory to store both the row and column complements of stripe S_k for both H, and A. This requires six n-dimensional arrays. The in-cell storage of the stripe relieves the host from supplying the same information to the cells in each iteration.

The other operations in the PCCG are essentially vector and scaler operations. A systolic cell similar to the ones used in ARRAY is added to the system to perform most of these operations. This cell is denoted by $\pi+1$ in Fig 7. In addition to multiplication and

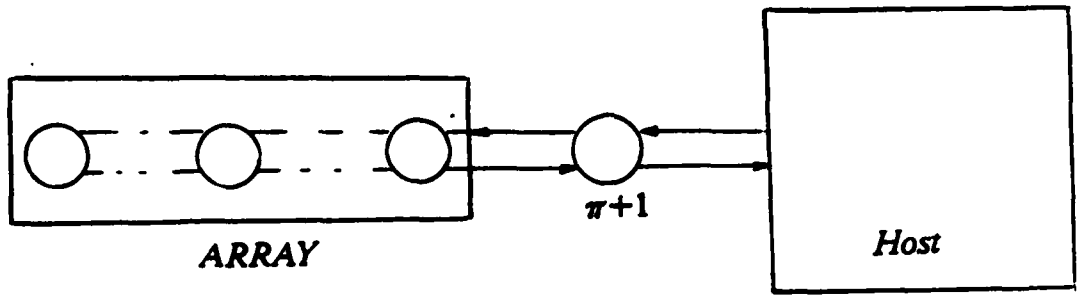


Fig 7 - A systolic system for PCCG

addition, cell $\pi+1$ should be able to perform division in either software or hardware. The time for division is not crucial because only two division operations are performed in each PCCG iteration, namely in steps 2.1 and 2.5. Also, cell $\pi+1$ is assumed to have local storage for a temporary array $SX_{\pi+1}$, and for the scalers α, β, γ_i and γ_{i+1}.

The host in Fig 7 is a general purpose computer which initializes the systolic cells and initiates and controls each iteration step. The vectors p_{i-1}, r_i and x_i reside in the host. At each iteration, the host supplies cell $\pi+1$ with the vectors p_{i-1} and r_i, and receives from it the vectors p_i and r_{i+1} and the scalers γ_{i+1} and α. It is also the responsibility of the host to compute x_{i+1} in step 3.1, and to check the value of γ_{i+1} for convergence.

	Step 1	Step 2	Step 3	Step 4	Step 5
Initial content of $SX_{\pi+1}$	h	y	y	b	h
Host → cell $\pi+1$	p_{i-1}		r_i		
cell $\pi+1$ → ARRAY	p_i		r_{i+1}	b	
ARRAY → cell $\pi+1$	y		b	h	
cell $\pi+1$ → Host	p_i	α	r_{i+1}		γ_{i+1}
Final content of $SX_{\pi+1}$	y	y	b	h	h
Execution in ARRAY	1.2		3.3	4.1	
Execution in cell $\pi+1$	1.1, 1.3	2.1	3.2	4.2	5.1
Execution in Host			3.1		
Execution time	$2(n+B_1+2\pi+3)$	1	$n+5$	$n+5$	1

Table 1 - Summary of communication and computation steps

The various computations assigned to each unit in the system and the data movement between units are summarized in Table 1. Each column in the table corresponds to one step in a PCCG iteration. Data movement is specified in the first six rows of the table, and computational activities are specified in the following three rows.

Given Table 1, it is possible to trace the execution of the system for each step in a PCCG iteration. For example, during the first step, the host sends the elements of p_{i-1} to cell $\pi+1$ at a rate of one element every two cycles. When cell $\pi+1$ receives the l^{th} element of p_{i-1}, namely $p_{i-1}[l]$, it computes $p_i[l]=h[l]+\beta p_{i-1}[l]$ (note that $h[l]$ is initially stored in $SX_{\pi+1}[l]$), and sends the result to both ARRAY and the host. ARRAY receives the elements of p_i and returns to cell $\pi+1$ the elements of the product vector $y=Hp_i$, at the same rate. Given that cell $\pi+1$ is busy executing step 1.1 only every second cycle, the idle cycles may be used for the computation of step 1.3. That is, whenever cell $\pi+1$ receives an element $y[l]$ from ARRAY, it accumulates its contribution to α, that is, it computes $\alpha=\alpha+y[l]*p_i[l]$, before storing $y[l]$ in $SX_{\pi+1}[l]$.

The last row in Table 1 gives, for each step, the number of systolic cycles required for the completion of the step. In order to simplify the table, it is assumed that each multiply, add, or divide operation terminates in one cycle. That is $p^+ = p^* = p^+ = 1$. By adding the execution times of the five steps, we conclude that PCCG iterations may be executed at a rate of one iteration every $4(n+\pi)+2B_1+10$ systolic cycles. Note that, every systolic cell in the system is doing useful work in almost every cycle. That is, for large n, the utilization of the system is almost 100 percent.

Using the WARP for the PCCG system

The Warp is a 10 cell systolic machine which have been developed at Carnegie Mellon University [1]. Each cell has a pipelined multiplier and a pipelined adder, with 5 stages each. In its current form, the Warp may not be used to implement the PCCG system described above, namely because it does only allow for a homogeneous mode of operation. That is all the cells should execute the same program. This requirement will be eliminated in the next version of the machine, namely the PC Warp.

The matrix/vector multiplication algorithm of Section 3 was implemented on the current Warp in order to test the suitability of Warp-like machines to sparse matrix manipulation. The W2 programming language [3] was employed and the W2 compiler was used

to generate the micro-code for the individual cells. Although, the computational results were correct, the timing results reported by the compiler indicates that the number of cycles required for execution is almost twelve times the theoretical number of cycles obtained in Section 3. This inefficiency is attributed to the following reasons:

1) In the current Warp, only one read and one write operation to/from local memory may be performed in any given cycle. In our algorithm, one of the multiplication operands and one of the addition operands have to be read from the local memory. Hence addition and multiplication may not be initiated in the same cycle.

2) The current Warp does not allow in-cell integer arithmetic. Hence any address generation required for array access (even those of the type *base +offset*) has to be done using the floating point addition unit. This source of inefficiency should disappear in the new PC Warp which includes an in-cell unit for integer arithmetic.

3) Inefficiency of indirect addressing: Although indirect addressing is supported by the Warp hardware, the task of managing this indirect mode by the compiler is difficult, especially when index arithmetic is performed in floating point with the result available after 5 cycles.

6. CONCLUSION

A complete systolic algorithm for the the solution of large sparse linear systems of equations is described. The algorithm may be implemented on a linear array of systolic cells attached to a host computer. Data flows regularly in the array and computation is distributed uniformly among its cells, thus leading to an almost perfect utilization of the resources. This perfect utilization, however, could not be achieved when the CMU Warp systolic machine [1] was used to implement parts of the system, namely because the cells in the Warp lack some of the basic capabilities that we assumed in our systolic array. Most of the lacking capabilities will be incorporated in future versions of the Warp, thus allowing for the implementation of an efficient solver for large sparse linear systems.

Acknowledgment

I would like to thank H. T. Kung and the Warp project group at CMU for their help during the implementation of the algorithms on the Warp machine. I would like also to acknowledge the supports of the Office of Naval Research and the Air Force Office of Research through the contracts N00014-85-K-0339 and AFOSR-84-0131.

References

1. M. Annaratone, E. Arnould, T. Gross, H. T. Kung, M. Lam, O. Menzilcioglu, K. Sarocky, and J. Webb, "Warp Architecture and Implementation," *Proc. of the 13th International Symposium on Computer Architecture*, 1986.

2. D. Evans, "The Use of Pre-conditioning in Iterative Methods for Solving Linear Equations with Symmetric Positive Definit Matrices," *J. Inst. Maths. Applics.*, vol. 4, pp. 295-314.

3. T. Gross and M. Lam, "Compilation for a High Performance Systolic Array," *Proc. of SIGPLAN 86 Symposium on Compiler Construction*, 1986.

4. H. Jordan, "A Special Purpose Architecture for Finite Element Analysis," *Proc. of the 1978 International Conference on Parallel Processing*, pp. 263-266 , 1978.

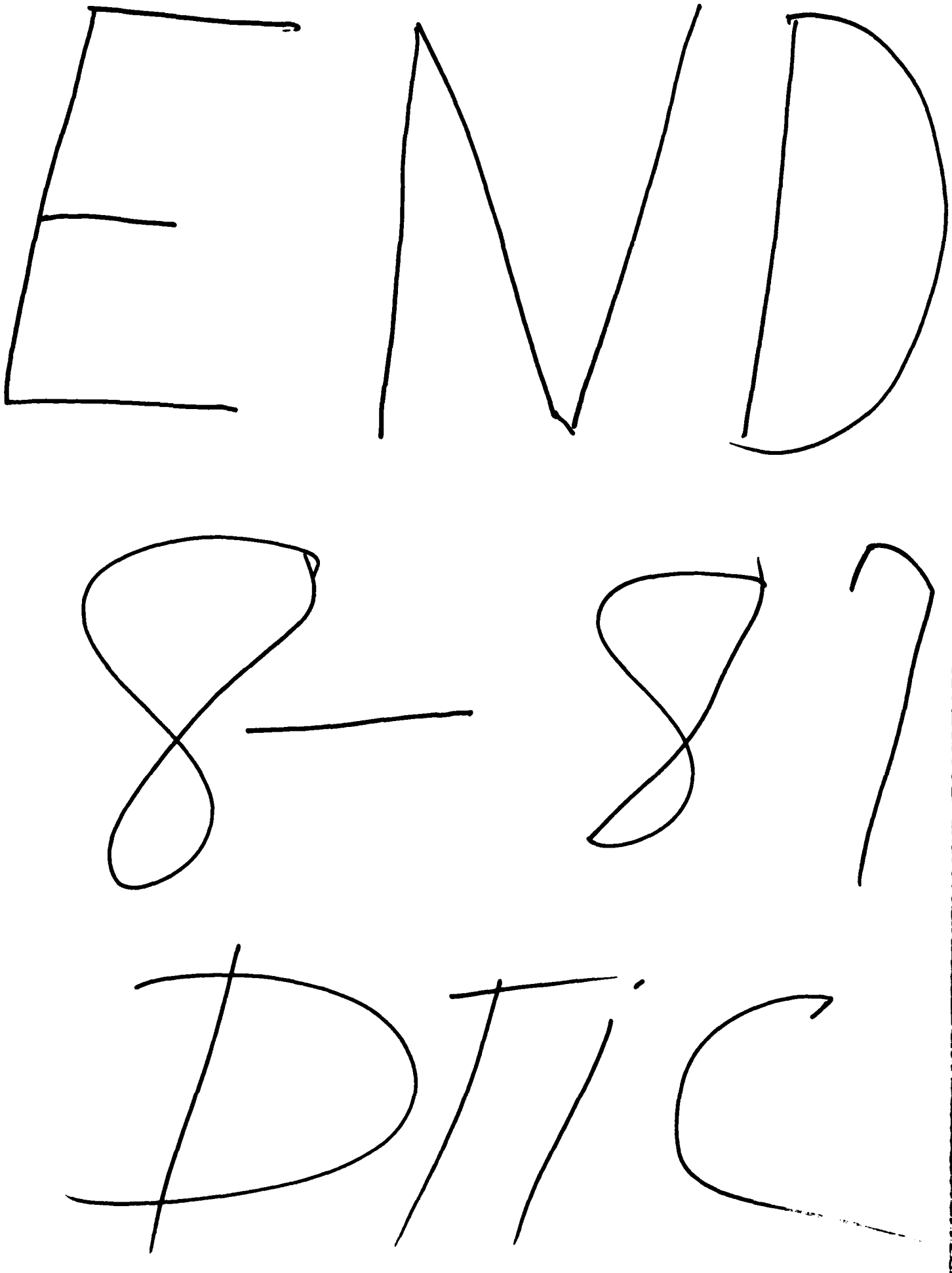
5. H. T. Kung and C. Leiserson, "Systolic Arrays for VLSI," in *Introduction to VLSI Systems*, ed. Mead C. and Conway L., Addison_Wesley, Reading Mass., 1980.

6. R. Melhem, "Parallel Solution of Linear Systems with Striped Sparse Matrices," Tech. Report. ICMA-86-91, January 1986. To Appear in Parallel Computing (1987).

7. R. Melhem, "Application of Data Drive Networks to Sparse Matrix Multiplication," *Proc of the 1986 International Conference on Parallel Processing*, 1986.

8. R. Melhem, "A study of the Stripe Structure of Finite Element Stiffness Matrices," Tech. Report ICMA-86-92, 1986. To appear in SIAM J. on Numerical Analysis (1987).

9. D. O'Leary, "Ordering Schemes for Parallel Processing of Certain Mesh Problems," *SIAM J. on Scientific and Statistical Computing*, vol. 5, no. 3, pp. 620-632, 1984.


END
8-87
DTIC